Timing

Author: Mark Szymczyk
Last Update: October 5, 2005

This article shows you how to accurately measure time on Mac OS X using the `Microseconds()` and `UpTime()` function calls.

Introduction

Every frame action games read the player's input, move objects that are in motion to their new positions, and draw the objects in their new positions. How much should you move the objects each frame? Assuming you store each object's velocity, you calculate how far to move each object by multiplying the velocity by the amount of time that elapsed since the previous frame. If an object's velocity was 100 units per second, and .1 seconds elapsed, you would move the object 10 units. Keeping track of the elapsed time keeps the game's physics independent from its frame rate on the player's machine.

There are two functions to help you calculate the elapsed time on Mac OS X: `Microseconds()` and `UpTime()`. Carbon applications can use these functions without having to do anything special. Cocoa programs that want to use these functions must add the Core Services framework to their projects. To use `Microseconds()` in a Cocoa program, you must include the file `Timer.h` in your program.

```
#import <CoreServices/Timer.h>
```

To use `UpTime()` in a Cocoa program, you must include the file `DriverServices.h`.

```
#import <CoreServices/DriverServices.h>
```

Microseconds()

The function `Microseconds()` returns the number of microseconds (1 million microseconds = 1 second) that have elapsed since the computer was turned on. By calling `Microseconds()` and comparing its result to the last time you called `Microseconds()`, you can determine how much time has elapsed.

`Microseconds()` returns its results in an `UnsignedWide` data structure. This data structure consists of two 32-bit unsigned integer values that the operating system treats as a 64-bit integer. The problem with using the `UnsignedWide` data structure to calculate the elapsed time is that you must subtract the previous call to `Microseconds()` from the most recent call. Subtracting two `UnsignedWide` values is difficult because the `UnsignedWide` value is split into two components. An easier way to calculate the elapsed time is convert the `UnsignedWide` value to a `double` value. To do the conversion take the upper 32 bits and multiply it by the value $(2 \wedge 32)$ and add the lower 32 bits.

```
double ConvertMicrosecondsToDouble(UnsignedWidePtr microsecondsValue)
{
      double twoPower32 = 4294967296.0;
      double doubleValue;

      double upperHalf = (double)microsecondsValue->hi;
      double lowerHalf = (double)microsecondsValue->lo;

      doubleValue = (upperHalf * twoPower32) + lowerHalf;
      return doubleValue;
}
```

After converting the UnsignedWide data structure to a double value, the math becomes easier. Just subtract the two double values to find the elapsed time.

```
double previousTime;

double CalculateElapsedTime(void)
{
      UnsignedWide currentTime;
      double currentTimeAsDouble;

      Microseconds(&currentTime);
      currentTimeAsDouble = ConvertMicrosecondsToDouble(&currentTime);

      double elapsedTime = currentTimeAsDouble — previousTime;

      // Convert elapsed time to seconds.
      double oneMicrosecond = .000001;
      double elapsedTimeInSeconds = elapsedTime * oneMicrosecond;

      previousTime = currentTimeAsDouble;
      return elapsedTimeInSeconds;
}
```

Looking at the code, you can see that I converted the elapsed time from microseconds to seconds. The reason I did this conversion is because I assumed the velocity for moving objects is expressed in units per second. If you want to express the velocity in units per microsecond, you can avoid doing the conversion.

UpTime()

The function `UpTime()` returns the amount of elapsed time since the computer started up. It returns the elapsed time in an `AbsoluteTime` data structure, which is a 64-bit integer. You can't directly do anything with the `AbsoluteTime` data structure. You must use one of the two conversion functions Apple supplies. The first function is `AbsoluteToDuration()`, which tells you the number of milliseconds (1000 milliseconds = 1 second) that elapsed since the computer was turned on. The advantage of using the `Duration` data structure is that the structure is a 32-bit integer. You don't have to convert it to a `double`. For games millisecond accuracy is usually sufficient.

```
double previousTime;

double CalculateElapsedTime(void)
{
        AbsoluteTime currentTime;
        Duration currentTimeAsDuration;

        currentTime = UpTime();
        currentTimeAsDuration = AbsoluteToDuration(currentTime);

        double elapsedTime = currentTimeAsDuration — previousTime;

        // Convert elapsed time to seconds.
        double oneMillisecond = .001;
        double elapsedTimeInSeconds = elapsedTime * oneMillisecond;

        previousTime = currentTimeAsDuration;
        return elapsedTimeInSeconds;
}
```

If you need more accuracy, use the second conversion function, `AbsoluteToNanoseconds()`. This function reports the number of nanoseconds (1 billion nanoseconds = 1 second) that elapsed since the computer was turned on. To make use of the nanosecond information, you must convert to `double` like we did with `Microseconds()`.

```
double ConvertNanosecondsToDouble(Nanoseconds* nanosecondsValue)
{
        double twoPower32 = 4294967296.0;
        double doubleValue;

        double upperHalf = (double)nanosecondsValue->hi;
        double lowerHalf = (double)nanosecondsValue->lo;

        doubleValue = (upperHalf * twoPower32) + lowerHalf;
        return doubleValue;
}
```

After making the conversion you can calculate the elapsed time.

```
double previousTime;

double CalculateElapsedTime(void)
{
        AbsoluteTime currentTime;
        Nanoseconds currentTimeAsNanoseconds;
        double currentTimeAsDouble;

        currentTime = UpTime();
        currentTimeAsNanoseconds = AbsoluteToNanoseconds(currentTime);
        currentTimeAsDouble =
                ConvertNanosecondsToDouble(&currentTimeAsNanoseconds);

        double elapsedTime = currentTimeAsDouble — previousTime;

        // Convert elapsed time to seconds.
        double oneNanosecond = .000000001;
        double elapsedTimeInSeconds = elapsedTime * oneNanosecond;

        previousTime = currentTimeAsDouble;
        return elapsedTimeInSeconds;
}
```

Conclusion

I have some code available to download that I wrote for the book Mac Game Programming. The code uses `Microseconds()` to calculate the elapsed time. Look at the functions `CalculateTimeStep()` and `ConvertMicrosecondsToDouble()` in the file `GameApp.cp`.